Journal of Ambient Intelligence and Smart Environments 6 (2014) 541–559
IOS Press


# Learning Distributed Deployment and Configuration Trade-offs for Context-Aware Applications in Intelligent Environments

Zubair Wadood Bhatti, Nayyab Zia Naqvi, Arun Ramakrishnan, Davy Preuveneers and Yolande Berbers

*iMinds-DistriNet, Department of Computer Science, KU Leuven, Belgium. E-mail: {zubairwadood.bhatti, nayyab.naqvi, arun.ramakrishnan, davy.preuveneers, yolande.berbers}@cs.kuleuven.be*

**Abstract.** The Internet of Things (IoT) is rapidly gaining ground as can be witnessed by the pervasive presence of the many things or objects around us that turn our surroundings into intelligent environments. These objects interact on a large scale in wired and wireless sensor and actuator networks using advanced communication protocols. Hence, IoT is an open ended and highly dynamic ecosystem with heterogeneous workloads and fluctuating resource availability. Distributed intelligence for smart objects and platforms is a vital enabling factor for IoT, but finding the best strategy to deploy and configure applications – especially those that require contextual intelligence – in a smart environment with dynamic and heterogeneous resource availability is far from straightforward.

Our experiments using context-aware applications for intelligent environments show that many resource and performance trade-offs exist and that current deployment schemes for these kind of applications are rough around the edges. We illustrate how a modular design philosophy for smart IoT applications enables a more optimal deployments. Furthermore, we present a methodology to inspect and learn the trade-offs of different deployment schemes of IoT applications in order to autonomously optimize their configuration. We validated our methodology on different use cases and scenarios, and the results demonstrate the feasibility of our approach to automate the efficient deployment of IoT applications in the presence of multiple conflicting Quality of Service (QoS) objectives and varying runtime circumstances.

Keywords: Deployment and configuration trade-offs, Pareto-exploration, Learning, Dynamic decision networks

## 1. Introduction

Unquestionably, the Internet of Things (IoT) is gaining momentum with a promise to significantly impact every individual and several aspects of their everyday life. Low cost wireless communication and efficient network performance lead to smart objects being capable of identifying, locating, sensing and connecting in the form of the Internet of Things, thus leading to new forms of communication between people and things, and things themselves. IoT is an open ended and highly dynamic network of uniquely identifiable fixed or mobile communicating objects. These heterogeneous objects, equipped with varying identification and tracking technologies, interact in wired and wireless sensor and actuator networks using advanced communication protocols. These resource constrained devices collect data, relay information to one another, process the information collaboratively, and take actions on behalf of their owners in an autonomic way to create intelligent environments. At the other end of the spectrum, the cloud provides a resource rich environment to host smart servers that run more demanding or long running tasks and applications. Areas such as smart homes and offices, smart health, assisted living, smart cities and transportation are only a few examples of possible application scenarios where smart servers are being used opportunistically to aid IoT in playing a vital role in our daily life. The unique selling proposition for end-users of combining cloud computing with mobile com-

munication is the ability to have data and applications running (partially) remotely in the cloud with easy access on the mobile.

This work focuses in particular on applications that exhibit smart behavior through situational awareness and on anticipation of human behavior where the relevant context information originates from within the mobile devices or other sensory devices. For the smart applications, a decision needs to be made whether context fusion and inference is best carried out on the device, or whether the raw sensor data is sent to the smart server to have (part of) it processed over there. Optimal deployment and optimization criteria are unclear for these open systems with fuzzy operating conditions. Multi-component application deployments that exhibit self-* properties are challenging to build. In this work we investigate how to change the behavior of a system to achieve a desired functionality, while maintaining a balance in conflicting optimization objectives, such as QoS and resource usage [44]. With the IoT being a large complex ecosystem, self-optimization in such environments shifts the focus from design and deployment of a single or a few elements operating autonomously to a network of autonomous elements [37].

Unfortunately, many of the existing IoT platforms are domain-specific prohibiting seamless interoperability of devices across multiple vertical domains such as Smart Home, Smart Health, Smart Transport, Smart Shopping, etc. The FP7 BUTLER project[1], in which this work is anchored, aims to address this concern by achieving a secure, context-aware horizontal architecture for the IoT by offering common functionalities on three types of platforms - *Smart Objects*, *Smart Mobiles* and *Smart Servers*. These platforms are categorized based on their intended use and prime capabilities, as described below:

- *Smart Object*: Devices with sensing or actuating capabilities to perceive or manipulate the environments in which they are embedded are categorized as Smart Objects. In most cases, these devices have limited processing and storage capabilities, and a primitive user interface to interact with (e.g. a switch). The typical examples are RFID-tagged objects, motion detectors, heating regulators, etc.
- *Smart Mobile*: Devices with sophisticated multimodal user interfaces, such as visual and audio cues along with traditional interfaces such as a keyboard, enabling user mobility through remote services. These devices usually have better computational resource capabilities compared to the smart objects. Typical examples include smart phones, tablets and smart TVs.
- *Smart Server*: When compared to the other two categories of devices, these systems have a large amount of computational resources and storage capabilities. They usually take care of aggregating and executing complex analysis of data collected from users and their environments through smart mobiles and smart objects. The typical examples are a local server, a remote cloud computing set-up or a hybrid computational deployment.

In this work, we challenge the hypothesis that using the cloud for all data storage and processing will always provide resource and performance benefits. We explore examples where the decision of deploying an application (or some of its subcomponents) on either the sensor, the mobile or in the cloud is not clear-cut. In our previous work [27,31] we identified that many resource and performance trade-offs exist, and we demonstrate that a modular application design philosophy helps to support optimal mobile cloud application deployments. The overall aim is to achieve a distributed intelligence by finding optimal distributed deployments and configurations of application components in the following way:

1. We use annotated component graphs to model application compositions at design time.
2. Pareto-curves are used to represent the optimization options for each (type of) platform. The resource optimization objectives are chosen w.r.t. to the QoS requirements and trade-offs between computation and communication.
3. We use dynamic decision networks for the runtime configuration and deployment to achieve the self-optimization capabilities of the system.

Our experiments show that with this combined approach, our framework is able to learn deployment trade-offs of smart applications for Intelligent Environments and capable of learning from earlier deployment or configuration mistakes to better adapt to the setting at hand. Section 2 compares our approach with related work. In section 3, we will present motivating use cases in the area of Intelligent Environments and elicit requirements from these scenarios. Section 4 provides an overview of the application components

[1] http://www.iot-butler.eu/

of our use case scenarios. The overall design-time and runtime methodology is discussed in section 5. We will illustrate the feasibility and the effectiveness of our approach through experimental evaluation in Section 6. In section 7 we conclude with our main findings and lessons learned, and list interesting topics for future research.

## 2. Related work

In this section, we discuss and position related work in the area of intelligent environments, activity recognition, mobile cloud computing and strategies for design-time and runtime optimization in the presence of uncertain operational conditions.

### 2.1. Intelligent environments

The overall aim of our work is to be able to predict and control the global system behavior resulting from self-optimization of the components deployed among the three *Smart Object, Smart Mobile* and *Smart Server* platforms. For intelligent environments, the dynamic deployment of software components in an IoT system has to take into account the resource characteristics of the application components and the deployment platforms in terms of processing power, bandwidth, battery life and connectivity [2]. Chen et al. [7] investigate challenges related to the fact that each platform has its own capabilities and limitations to achieve certain Quality of Service (QoS) requirements. They present a context-aware resource management approach for service oriented applications with the ability to handle the inherent service and network dynamics and to provide end-to-end QoS in a secure way. In [45], Yang et al. explore the distributed recognition of human actions using wearable motion sensor networks. They propose a distributed sparsity classifier and demonstrate an example of an multi-optimization concern of the classifier being able to conserve sensor energy for communication while preserving accurate global classification.

### 2.2. Activity recognition

Analysis of physical fitness and several health monitoring techniques revolve around the inference and prediction of human behavior. Accelerometer data helps to analyze the human behavior in an effective way. With proper processing of this raw data, a bunch of human activities can be inferred [32,23,21]. Apart from inferring human activity, this data is useful to analyze the patterns [1] in human behavior in order to predict human activity and achieve a whole new kind of health monitoring systems [9,28,39]. Activity learning [38] is a growing research field to achieve a true autonomous ecosystem. As today's smart phones come with an increasing range of sensing, communication, storage and computational resources, they can play a special role in realizing effective human-centric systems. Mobile devices have built-in sensors to sense the situation of the users. As such, they are important for fetching the context data, but the size of the acquired context data varies, depending on the application's objectives.

### 2.3. Mobile cloud computing

Extraction of a specific context can be computationally expensive and problematic in mobile and uncontrolled environments due to the shortage of resources for computation, data storage, network bandwidth, and battery capacity. Furthermore, continuous learning on the basis of this updated context data is almost impossible to be carried out in a mobile device due to the aforementioned limitations. Contrary to mobile devices, cloud computing provides plentiful storage and processing capabilities. That is why mobile applications are being built based on web standards to offload computation to the powerful cloud resources.

Several surveys [12,16,30] have been conducted to analyze the state-of-the-art in the mobile cloud computing paradigm. Dinh et al. [12] survey the state-of-the-art in mobile cloud computing and discuss the challenges to embrace cloud computing for mobile applications. QoS is an open issue in mobile cloud computing research. CloneCloud addresses this issue by cloning the entire set of data and applications from the smart phones to the cloud and selectively execute operations on the clone. However, for the continuous nature of context data in the setting of human behavioral analysis, the CloneCloud approach fails in a way that the dataset is not predetermined and needs to be processed in real-time for healthcare applications. Cloudlets [34] is another approach to achieve QoS but it does not provide an answer for the distribution of processing, storage, and networking capacity for each cloudlet. How to manage policies for cloudlet providers to maximize user experience while minimizing cost, security and trust are also open issues in order to adopt it for practical systems.

Mobile healthcare monitoring techniques need to be more energy efficient to improve their practical use in a real world setting. Research in [22,26] suggests that cloud computing can potentially save energy for mobile users. However, not all applications are energy efficient when migrated to the cloud. Although process offloading to the cloud can be beneficial in terms of processing performance, the distribution of tasks is never clear-cut when considering multiple QoS trade-offs. Certain healthcare applications are pretty lightweight, but others require a lot of computational effort (e.g. for prediction) or require analysis of large amounts of data (e.g. for pattern analysis).

Our framework addresses this concern by identifying the deployment trade-offs with respect to resource and performance for offloading data and computation. If the decision is affirmative, the required functionality is executed as a composition of loosely coupled services on the cloud. Otherwise lightweight component-based equivalents of these services are executed on the mobile.

### 2.4. Design time optimization

Deployment of application components in dynamic intelligent environments is often a multi-objective optimization problem [11,25]. The optimization objectives often conflict with each other, such that one objective can only be optimized at the cost of another and a single solution that optimizes all criteria does not exist. An example is Quality of Service (e.g. in terms of performance or responsiveness) and energy consumption of mobile applications where the QoS can only be increased at the cost of a higher energy consumption. Therefore, resource trade-offs [19] have to be made. Pareto optimization [6,46] is a technique that identities a set of Pareto-optimal solutions involving more than one objectives to be optimized simultaneously.

Tesauro et al. [42] presents utility functions as a way to enable a collection of autonomic elements to continually optimize the use of computational resources in a dynamic, heterogeneous environment. Later work by Deb et al. [10] study how utility functions can be applied to achieve self-optimized deployment of high performance computing scientific and engineering applications in highly dynamic and large-scale distributed computing environments. Utility functions are also being applied in the cloud computing space [18, 20] where they are used to manage virtualized computational and storage resources that can scale on demand. The problem with utility functions is that their definitions require a fair amount of domain-specific knowledge to be effective.

Sawyer et al. [35] present a constraint programming based approach for configuring wireless sensor networks (WSNs). Functional and Quality of Service requirements are modeled as a set of goals and the dependencies between them. Different configurations (i.e. hardware and software) may operationalize the same goals in different ways and a constraint solver is used to find the optimal configuration. In contrast, we use a dataflow-centric approach. By extracting a dataflow model of the application and mapping it onto an abstract model of the physical system we find the optimal solutions for the deployment of the application and configuration of the hardware.

The problem with static approaches, such as utility functions or Pareto-optimal solutions, is that the Internet of Things is a dynamic open ended ecosystem of heterogeneous resources, where external factors and uncertain circumstances influence the optimality. Furthermore, the applicability of the above learning approaches in an Internet of Things environment is usually hampered by the time and computational resources required to find a feasible or better solution. To address this concern, we aim to explore the feasibility of finding reasonable results in a reasonable amount of time by combining Pareto-optimization with run time optimization techniques.

### 2.5. Runtime decision support for optimization under uncertain operational conditions

Ubiquitous systems have to take into account uncertainty in context data at runtime. In these systems context sources are dynamic in nature. They can disappear and re-appear at any time and context models change to include new context entities and types. The properties of context sources and context types can change randomly and the uncertainty can vary too.

The problem with utility functions is that their definitions require a fair amount of domain-specific knowledge to be effective. To address this challenge, reinforcement learning is often considered to automatically infer optimal deployment strategies. Tesauro [40, 41] explored reinforcement learning for an online resource allocation task in a distributed multi-application computing environment with independent time-varying load in each application. Similar work was proposed by Vengerov [43] using reinforcement learning in conjunction with fuzzy rulebases to achieve the desired objective. However, long training times is a reoccur-

ring concern that often outweighs the potential benefits of reinforcement learning. Fenton and Neil [14] have used Bayesian networks for predictions of the satisfaction of non-functional aspects of a system. Esfahani et al. [13] employ fuzzy mathematical models to tackle the inherent uncertainty in their GuideArch framework while making decisions on software architectures. Dynamic configuration of service oriented systems was investigated by Filieri et al. [15]. In contrast to our model, they used Markov models to investigate the decision making under uncertainty and quality requirements. Our approach focuses on evaluating the role of QoC for decision support in highly dynamic and open ended ubiquitous systems. We have a different perspective to tackle the challenge of ever changing contexts and making decisions in time based on that context. We have used probability reasoning with Bayesian networks and Decision networks [33] to tackle the real-time decision problems under uncertainty and QoC requirements of the users. Moreover, we focus on runtime aspects of the uncertainty of the context data and their impact on the actions of these systems leveraging DDNs. We model a real-time ubiquitous system that dynamically changes over time. Its context and QoC requirements for each user also evolve over time. Our model aims to learn from the previous decisions in order to improve the quality of the decisions and actions by our system.

## 3. Intelligent environments: Motivating use cases and requirements

Recent advancements in ICT have helped to enable people with special needs not only to lead socio-economically independent life-styles but also a possibility to make full range of choices in all life spheres with social and environmental responsibilities [8]. In this section, some motivating scenarios from the Ambient Assisted Living (AAL) and Ambient Intelligence (AmI) domains have been used as prototypical examples of IoT applications in smart environments. One of the primary goals is to support users with infrastructure attuned to their senses and provide services to make them more autonomous and yet feel in-control of their situation.

### 3.1. Motivating use case scenarios

The first motivating use case deals with monitoring a users' general well-being and the detection of alarming situations, whereas the second use case focuses on being more socio-environmentally responsible by being energy-aware of the devices people use in their everyday life.


Fig. 1. Object recognition in the mobile Smart Lens application using the Vuforia library

#### 3.1.1. Use case 1: Accelerometer-based fitness monitoring and fall detection

Physical wellness and health are highly inter-linked as mobility is seen as an essential decisive factor to maintain an altogether independent living. A detailed account of assisted living technologies and functions have been outlined by Sun et al. [36]. Ensuring the safety and security of the user with the help of alarms, monitoring the health and well-being of the user, and the use of interactive and virtual services to help support the user are just a few of them. Hence, in the first use case, the mobility of the user is being monitored by inferring the physical activity of the user (standing, walking, number of steps taken, etc.). The system detects a fall in a smart way by not only relying on data provided by accelerometers, but also by incorporating knowledge about the location of the fall to infer the likelihood of the fall (for example, to reduce false positives due to dropping yourself in a chair) and to notify the caregiver in case an emergency situation has occurred.

#### 3.1.2. Use case 2: Energy awareness about everyday appliances using augmented reality

Electrical energy usage is one of the growing concerns nowadays due to its economic and environmental impacts. Hence, the second use case considered in this paper is an augmented reality mobile application (called *Smart Lens*) that enables the user to be effortlessly aware of the power consumption of any specific appliance to motivate themselves and make an informed decision to save energy. Fig. 1 illustrates

the Smart Lens application and the Vuforia[2] library in progress of identifying the target object based on selected features (green dots in picture on the right). In addition to the near-real-time power consumption of the appliances, the Smart Lens application allows the user to get a detailed overview of the impact of any particular device compared to others in their electricity bill. Also, by storing the historic data it allows the user to monitor the performance of a device over time and determine whether it is time to replace a device with a more energy-efficient alternative.

### 3.2. Objectives

For the first use case, we use an accelerometer based mobile wearable solution for monitoring the physical activities of the user and for detecting falls. The design choice to use the tri-axial accelerometer readily embedded in the smart phones is based on the requirement for a pervasive, non-intrusive and mobile sensing unit. Also, due to their embodiment within the phone, it eliminates the need for radio-communication of the data. In the case of fall detection, the objective is to realize an effective, automated recognition and alarm system for emergency situations in a smart home environment with emphasis on near real-time analysis and minimal false positives.

For the second use case, the users can get the real-time update on power usage of any device at home/office environment using the Smart Lens application by pointing their mobile device at the specific device and capturing its image with the embedded camera. The Smart Lens application (*Smart Mobile*) supports markerless recognition of the common commercial devices for activities of daily living in real-time by extracting natural image features and comparing snapshots with a set of target images already available in the database. This application utilizes the location information of the user and the appliances to reduce the search space of objects to recognize and make the comparison smarter. The instantaneous power consumption details of the devices are gathered by the smart plugs (*Smart Objects*) and stored in the servers along with the database of the list of devices and their images (*Smart Server*).

Note that the work presented here does not necessarily focus on further improving the recognition capabilities of the more complex activity monitoring systems or improving the performance of image recognition algorithms. Instead, it aims to explore the trade-offs for providing practical Intelligent Environment solutions with reasonable performance on the one hand, and having considerably low resource consumption by learning how to optimize the distributed deployment and configuration of the individual components.

[2]https://developer.vuforia.com/resources/sdk/android

### 3.3. Requirements elicitation

The requirements elicitation is an important phase in the software engineering process for the development of smart applications [29]. The functional and non-functional requirements listed below were elicited from discussions between various stakeholders – from developers to end-users – in the systems and applications. A key observation was that the context in which the application is being consumed becomes an integral part of how a system will be used. Hence, the flexibility and the ability of the system and application to adapt to user preferences were identified as key features. How the mapping of user preferences onto concrete configurations is implemented and exposed in the user interface is application-specific and beyond the scope of this work.

#### 3.3.1. Functional requirements

Both the use cases have modular building blocks to manage the data and control flow to enable a flexible distributed deployment. The main functional system requirements for both the use cases are:

1. **Use case 1**: The system can model, learn, classify and predict the current physical activity of the user and quantify the physical intensity of the activity.
2. **Use case 1**: The system can record all the relevant information for offline learning and analysis by health care professionals.
3. **Use case 1**: The system can detect the physical state of the user immediately before and after the fall impact to assess whether assistance is needed in order to eliminate/minimize the false alarms.
4. **Use case 2**: The system should be able to identify the electronic appliance, preferably with a markerless approach.
5. **Use case 2**: The system should be able to show the energy consumption of the appliances.

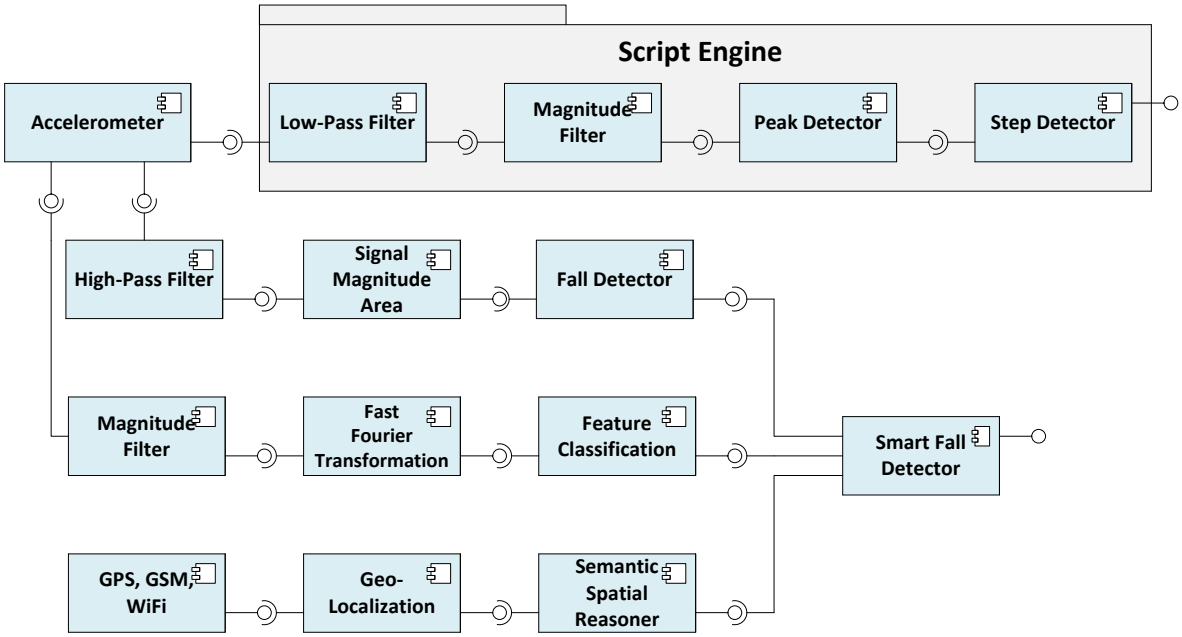


Fig. 2. UML component diagram of the activity monitoring and fall detection application

*3.3.2. Non-functional requirements*

More importantly in the frame of this work are the non-functional requirements that relate to the Quality of Service (QoS) properties that the system should achieve. We summarize the main requirements for both the use cases below:

1. **Use case 1**: The system should detect the impact in real-time to facilitate immediate notifications to appropriate care givers or emergency response teams.
2. **Use case 2**: The system should minimize the number of comparisons with the stored samples in order to recognize the appliance efficiently.
3. **Use case 1 and 2**: The system must be adaptive at runtime to optimize the resource consumption of the application (e.g. by offloading the processing when the mobile's battery is running low).
4. **Use case 1 and 2**: The system should be able to capture the real-time context of the user and his environment.

With several distributed deployments of the application components and different configurations per component, many optimization trade-offs exist and the challenge is to find and analyze them in an open ended and dynamic IoT ecosystem of *Smart Objects*, *Smart Mobiles* and *Smart Servers*, each characterized by varying sensing, communication, computation and storage capabilities.

## 4. Application components

This section briefly describes the modular component-based design of our applications, which simplify the re-deployments and reconfigurations significantly.

### *4.1. Building blocks of the fitness monitoring and fall detection use case*

Fig. 2 provides an overview of the composition of the components for the first use case.

– **Accelerometer**: This component generates sensor data at a certain sampling rate. This data is consisted of a triplet of X,Y,Z values.
– **Low-Pass Filter**: A 'moving average' component is used as a low-pass filter to track the mobility of a user, e.g., walking or running. We capture the acceleration peaks arriving at a frequency of maximum 5Hz (i.e., max 5 steps per second) to remove high-frequency noise.

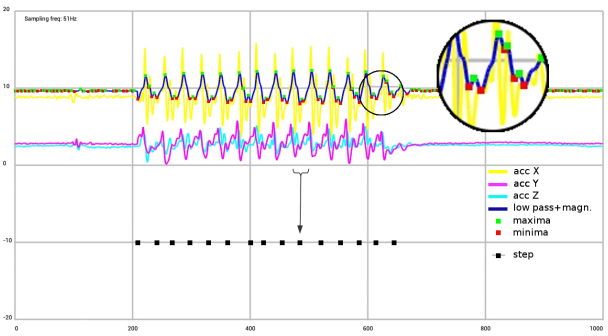


Fig. 3. Peaks in magnitude signal and detected steps

- **Magnitude Filter**: In many cases, we do not know how the accelerometer is oriented at the offset of the motion activity. Furthermore, the orientation of the sensor is subject to change while moving around. Therefore, we carry out the signal analysis on the overall magnitude of the acceleration signal.
- **Peak Filter**: This component extracts a pattern of maxima and minima in the time domain of the acceleration signal to analyse the steps taken (see Fig. 3).
- **Step Detector**: This component identifies the same peak for every step in order to correctly count the number of steps and to differentiate between standing still, walking and running (i.e., the peak rate) (see Fig. 3).
- **High-Pass Filter**: For fall detection, we are interested in sudden and high-frequency changes of the acceleration signal, both in amplitude and orientation. This component implements a high-pass Finite Impulse Response (FIR) filter to extract these features.
- **Fall Detector**: This component analyzes the signal magnitude area (SMA) of the high-frequency part of the acceleration signal, and identifies a fall if this feature passes a certain threshold.
- **Fast Fourier Transformation**: This component takes a time domain signal and converts it into a frequency domain signal. It is used for feature extraction on the magnitude signal and provides input for activity training and classification.
- **Feature Classification**: Whereas the previous components mainly condition and extract useful features from the accelerometer data, this component builds decision models through training (leveraging with Weka machine learning library) to classify the activity of the user.
- **Script Engine**: Rather than deploying separate and dedicated components to implement some of the above features, this component implements a lightweight scripting environment for simple mathematical computations that can easily replace and combine several components.
- **Geo-localization**: This component uses a variety of algorithms and filters to compute in-doors localization based on signal strength and time of arrival from ranging nodes to localize the users and associated objects in real-time.
- **Semantic Spatial Reasoner**: A Parliament based semantic reasoner is used to map the geo-location information from the previous component to meaningful semantic descriptions (e.g., kitchen, couch in the living room, etc) that are better understandable for the end users.
- **Smart Fall Detector**: A component to co-relate the falls and the semantic location of the user. It is a probabilistic model that calculates the risk and analyses the emergency situation to notify the caregiver.

Evidently, activity recognition and the smart fall detector are the most significant as well as the most resource consuming task of the system. This system utilizes the built-in tri-axial accelerometer to do opportunistic sensing. Although the energy efficiency of the accelerometers has increased over the years, continuous sensing at high frequencies and onboard processing of these data streams have proven to rapidly drain the battery of mobile devices [24]. Hence, the major challenges with sensing are to determine the optimal sampling frequency and extracting the suitable features depending on the other available context information.

We use an implementation of a subset of the Weka library that runs both on Android and in the cloud to classify different types of activities (*standing still*, *walking* and *running*). The Waikato Environment for Knowledge Analysis (Weka) [17] suite[3] is an open source data mining catalog of machine learning algorithms implemented in Java. It provides techniques for data pre-processing, clustering, classification, regression, visualization, feature selection, training and classification. The idea is that (1) we compute the magnitude of the raw accelerometer data (without applying any low-pass or high-pass filtering), (2) apply a Fast Fourier Transformation, (3) compute the magnitude of

[3]http://www.cs.waikato.ac.nz/ml/weka/

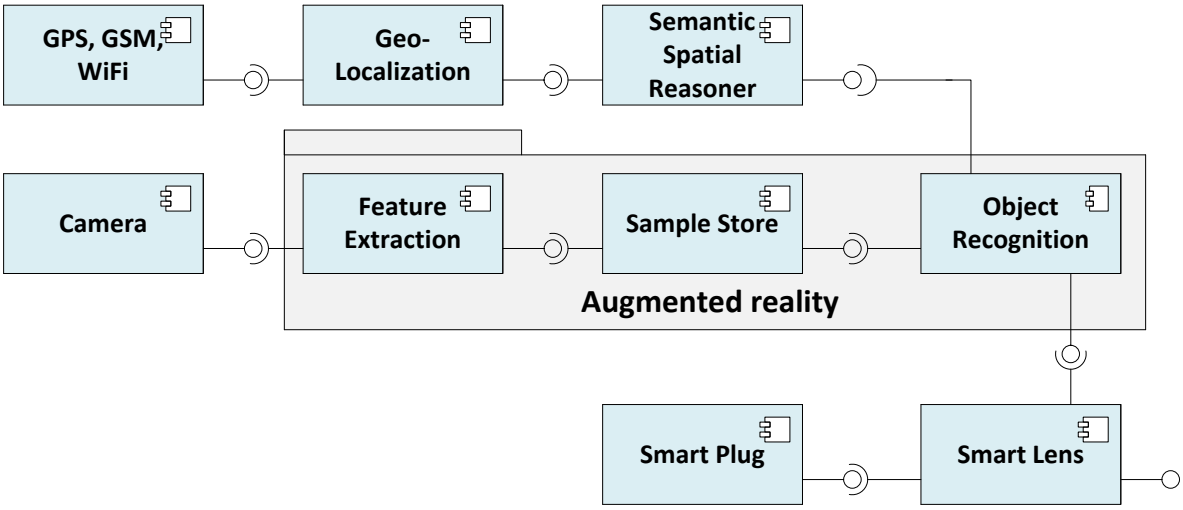


Fig. 4. UML component diagram of the energy awareness application components

the signal in the frequency domain, and (4) use those coefficients as features for training and classification of the user activities.

Despite using opportunistic sensing, precise labeling of the training data is a significant challenge as most models require extensive training data for good classification accuracy. This problem worsens as the scale of this system increases. Therefore, the primary motivation of this paper is to build a scalable activity recognition system for mobile devices with intelligent hybrid sensing, inference and learning that can leverage the resource rich environment of the cloud.

### *4.2. Building blocks of the mobile energy awareness use case*

Fig. 4 provides an overview of the composition of the components for the second use case.

- **Camera**: A continuous stream of images will be produced by the camera for further processing and device recognition.
- **Geo-Localization**: We have re-used the components described for use case 1 to get the fine grained semantic location of the user to short list the most probable list of appliances that are shown by the camera.
- **Augmented Reality**: We have utilized the Vuforia library to recognize the everyday appliances of the users in their smart home environment. It performs two sub-tasks: (1) extracting features from the captured images and (2) comparing it with the existing database to recognize the device of interest. The semantic location information from the localization component is used to reduce the search space of the reference images.
- **Smart Plug**: In order to be aware of the energy consumption of the everyday user appliances, we have utilized off-the-shelf smart plugs that measures and updates the real time power consumption of these devices through web-services.

## 5. Overview of the methodology to learn deployment and optimization trade-offs

The previous section gave an overview of the different building blocks used in our two use case applications, explaining the different software components in detail. The large number of parameters associated with the deployment configurations for these applications make it nearly impossible for developers to fine-tune them manually. Therefore, automated deployment and optimization are necessary. This section presents a methodology for the automated deployment and configuration of these component based applications.

Carrying out a detailed cost benefit analysis for each configuration and adaption decision at runtime causes a large overhead. However, this overhead can be reduced by balancing the offline and runtime efforts of making dynamic deployment decisions. This approach

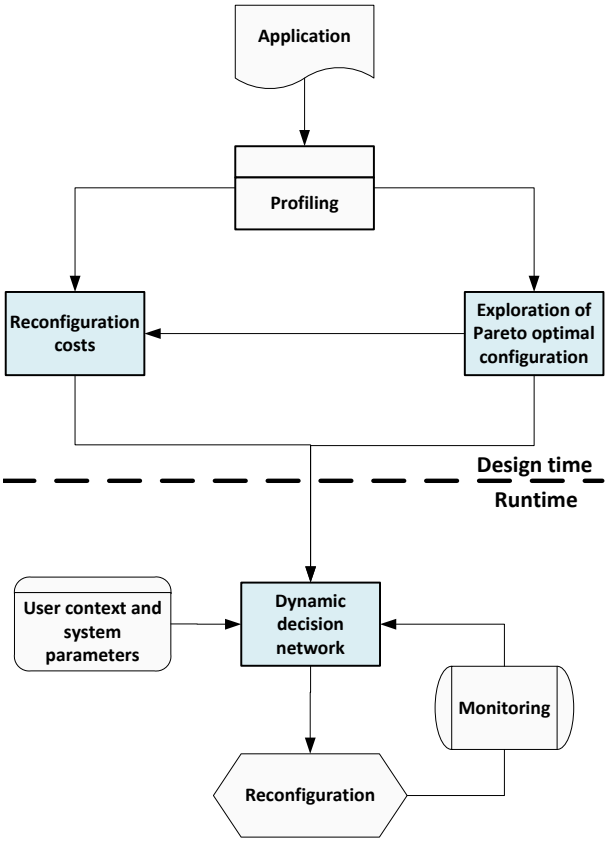


Fig. 5. Overview of the approach illustrating the offline and runtime phases

consists of two phases; a design-time phase and a runtime phase. Fig. 5 gives an overview of the methodology highlighting the important steps. At design time application components are profiled and optimal configurations for a set of possible runtime situations are computed along with the costs of switching between these configurations. At runtime these pre-computed configurations are used as per the context of the user and other runtime parameters of the system.

### 5.1. Design time phase

In Fig. 6, we give an overview of the offline exploration phase to maximize the preprocessing of deployment and configuration decisions. Our applications are component based, and this black box development methodology allows applications to be represented as annotated component graphs. These are directed graphs where the nodes represent the application components and the edges represent the dataflow between these components. The graphs are used for the exploration of the Pareto-optimal deployments and configurations. Once the Pareto-optimal configurations have been defined, the reconfiguration cost matrix is constructed for these selected configurations. Later on, the runtime system will use the Pareto-optimal configurations and cost matrices in order to carry out self-optimization decisions at runtime.

#### 5.1.1. Profiling the application component

During the profiling phase, the component graph is annotated with metadata regarding the resource requirements of each components and the dataflow between them (i.e. the CPU time, energy consumption and memory requirements for the components and the amount of data being transferred along the edge). The metadata is collected by profiling the execution of components on (a subset of) the different platforms, i.e. the *Smart Objects*, *Smart Mobiles* and *Smart Servers*. The profiling phase consists of the following steps:

1. Use the component model of the application and identity the data flows (similar to the one shown in Figure 2). The data flow graph acts as a skeleton for the annotated component graph.
2. Instrument the communication interfaces of components to measure the amount of data transferred between components.
3. Run every component of the application on all the different platforms possible, profiling its execution time, energy consumption and data transferred between components, each time.
4. Calculate the memory requirements of every component by monitoring the changes in stack and heap sizes, as components are added and removed from the platform.
5. Repeat steps 3 and 4 over a range of component configurations (e.g. a different sampling rate) and/or simulated inputs (e.g. accelerometer traces of different activities and individuals).

The only hard constraints for these kind of applications are that the accelerometer, GPS and camera components can only execute on devices with such sensors. Some components have configuration options that affect their resource costs and requirements. For such components we annotate the component graph with metadata for a discretized range of parameter options, i.e. the CPU time and energy consumption values for the supported sampling rates. Adding all this meta-data

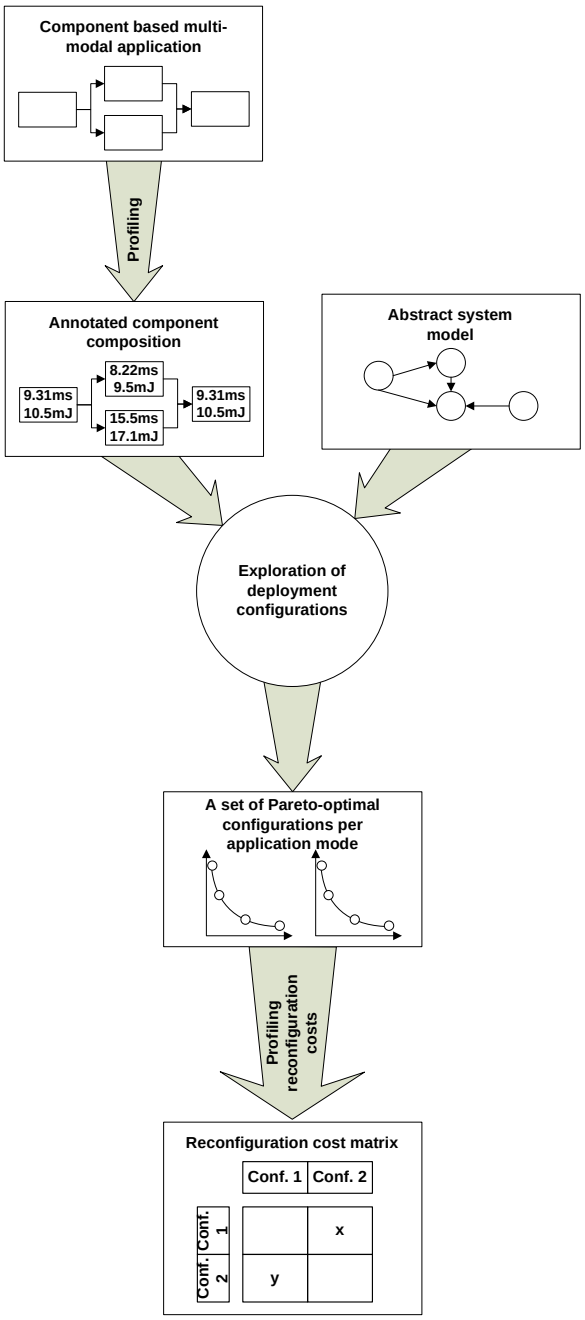


Fig. 6. Overview of the offline exploration phase

to the data flow graph generates the annotated component graph of the application and we use it as an intermediate model for exploring Pareto-optimal deployment trade-offs at design time.

| Investment | Volatility | Rate of return |
|---|---|---|
| **A** | 0.50 | 7.0% |
| **B** | 0.45 | 7.5% |
| **C** | 0.51 | 8.5% |
| **D** | 0.65 | 9.6% |
| **E** | 0.63 | 9.8 |
| **F** | 0.70 | 11.1 |

Table 1
Maximization problem with multiple optimization criteria

### *5.1.2. Pareto exploration*

We model the problem of deploying an application to a heterogeneous network of (re)configurable nodes *Smart Objects*, *Smart Mobiles* and *Smart Servers* as a constraint-based optimization problem. The details of expressing software deployment on hardware resources and how we explore the Pareto-optimal set of solutions with a CPLEX based solver[4] are described in our previous works [4,5].

A set of solutions is *Pareto-optimal* if every solution in the set is better than all other solutions according to at least one functional or non-functional criterion. For example, Table 1 refers to a hypothetical scenario with six investment possibilities that have varying volatility and rate of return. We want the highest rate of return with the least volatility but we have to make a trade-off. Investments A and D can be eliminated from the set because they are not Pareto-optimal. Also note that investments C and E are not the best in any optimization objective (volatility, rate of return), but they are Pareto-optimal. Although we are mainly interested in deployment and configuration of component based applications, we use this example to offer a better understanding of Pareto exploration with multiple optimization criteria.

Eliminating deployment and configuration options that are not Pareto-optimal reduces the search space for the runtime reconfiguration decision from all possible configurations to the set of Pareto-optimal configurations. For example, consider the first use case in section 3 which consists of 13 components, 11 of these components may be deployed on three different platforms. Deploying more components on the *Smart Objects* and *Smarts Mobiles* stresses these devices in terms of memory, computational power and battery consumption, whereas deploying more components on the *Smart Server* adds to the communication costs.

[4]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

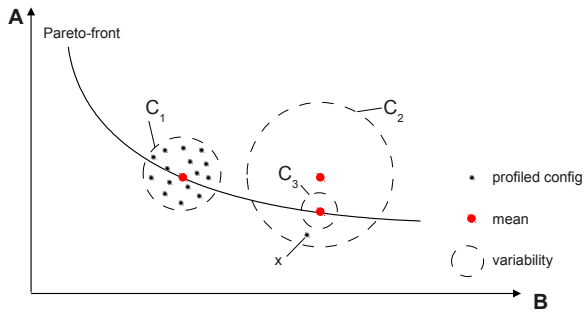


Fig. 7. Variability in the profiled configurations

Moreover, some of the components have different operational modes (e.g. the sampling frequency for the accelerometer component) again with different costs. All of these permutations combined form a large solution space, exploring it at runtime would incur a huge overhead. Therefore, at design time a set of Pareto optimal configurations if computed (in order to reduce the solution space of the runtime phase).

Finding optimal deployment configurations involves exploring a large space of multi-dimensional criteria, and this is computationally expensive. Therefore, we find a set of Pareto-optimal solutions at design-time and use it to limit the search space at runtime. In order to explore multi-dimensional Pareto-optimal surfaces, we model the problem with parameterizable constraints. These parameters are then iteratively varied over a discretized range, invoking the solver each time to find a point on a Pareto surface. For example, a Pareto curve for energy consumption versus performance (QoS) for the step counting algorithm is explored by iteratively finding minimum energy solutions for different performance constraints.

Depending on the number of simulations, finding solutions with the the CPLEX solver usually takes several minutes on a single machine. However, as there are no dependencies among the different invocations of the CPLEX solver, we can speed up this process by initiating parallel invocations of the CPLEX solver on a cluster of machines. This guarantees the feasibility of the approach for larger applications with many more components and configuration alternatives.

#### *5.1.3. Exploring reconfiguration costs*

The Pareto front provides configurations that are optimal for a given runtime situation. However, all these configurations are not feasible or the most optimal for all runtime situations. Therefore, switching from one Pareto-optimal configuration to an other is often required. Let us hypothetically consider an application with two runtime situations; *context A* and *context B* and two Pareto optimal configurations; *configuration X* and *configuration Y*. Configuration X is cheaper than configuration Y. However, it is only valid for context A and switching between the configurations requires N mAh. Now if the system is in the configuration Y and the context changes from B to A, the runtime system has two options; either to stay in configuration Y or the switch to the cheaper configuration X. The decision of whether to switch to configuration X or to stay in configuration Y depends on how long the context A will last. If it lasts only for a very short period, the cost of switching is not recovered. The final decisions of whether or not to switch between configurations is done at runtime, however, a table representing the costs of switching between different configuration is compiled at design time.

Some components have stochastic non-functional performance properties (see Fig. 7). For example, the communication throughput of a wireless node could be affected by external factors (e.g. interference). To define the Pareto-fronts (or Pareto-curves) for closed real-time systems the worst case execution estimates are usually taken after profiling to define the Pareto-points. Given that the IoT ecosystem is quite heterogeneous and open ended in nature, pursuing such a pessimistic approach will easily lead to undesirable solutions. Therefore, we define the Pareto-points based on the most likely execution values. However, to still be able to assess the impact of a worst case execution scenario for a particular deployment and configuration (i.e. a specific Pareto-point), we incorporate the likelihood distribution of the profiled execution values in each Pareto-point leading to a Pareto-front (i.e. a set of Pareto-optimal solutions) with some degree of variability. A reconfiguration cost matrix is constructed by profiling the costs of reconfigurations and redeployments of components. For example, the cost of activation/deactivation of a component, establishing a local/remote component -to-component communication channel and transferring the state of an active component over a communication network. The size of this matrix is $O(N^2)$ where $N$ is the number of possible configurations. As $N$ can be become large, only the Pareto-optimal configurations are considered for reconfigurations.

### *5.2. Runtime phase*

At runtime dynamic decision networks (DDNs) [3] are used for the reconfiguration and redeployment de-

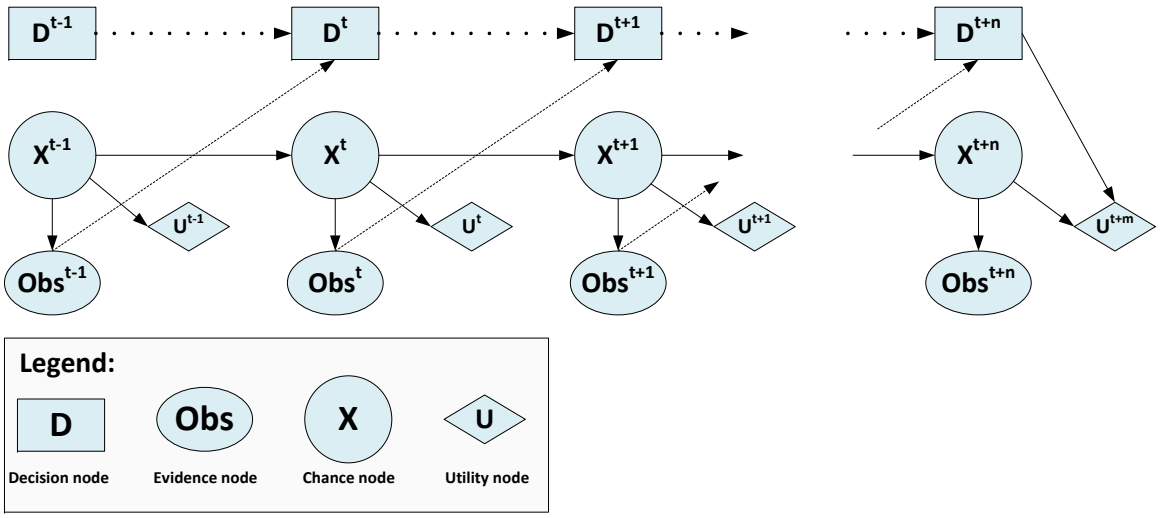


Fig. 8. Structure of a general dynamic decision network [33]

cisions. DDNs observe the user context and various system parameters (such as, remaining battery and network connectivity) to make the redeployment decisions using the meta-data generated in the design time phase i.e. the Pareto optimal configurations and the cost of switching between these configurations. In this section, we first describe the basic concept of dynamic decision networks and then show how they are applied.

*5.2.1. Dynamic decision diagrams*

Model-driven engineering and runtime models play a crucial role in tackling the influence of uncertainty in data. A key issue in this approach is keeping the runtime models synchronized with the changing system. Uncertain attributes can be described using probability distributions derived by analyzing historical attribute values. These methods can take advantage of probability theory and statistics that helped solve stochastic problems in the past. Probabilistic reasoning systems use network models to reason with uncertainty. Probabilistic reasoning allows the system to reach rational decisions even when complete information is not available.

Knowledge about runtime uncertainty can be captured by a data structure for probabilistic inference called a Bayesian network (BN) and can be extended with temporal decisions, their utility and chance nodes to become a DDN. Bencomo and Belggoun [3] have advocated to use DDNs to deal with the runtime uncertainty in self-adaptive systems. DDNs can be used to model the decision support system that passively monitors and predicts the environment over time to take correct actions while considering any preferences. We present a mathematical model supported by DDNs as a solution to address the uncertainty in the context data and its quality while taking into account QoS requirements. A BN is a Directed Acyclic Graph (DAG) represented by a triplet (N, E, P), where N is the set of chance nodes, E is the set of arcs to represent causal influence of the chance nodes and P is the conditional probability distribution for each chance node.

A Decision Network is a BN that also includes a set of decision nodes and utility nodes. The utility nodes express the preferences among possible states of the world in terms of a subset of the chance nodes and the decision nodes. A probability-weighted expected utility is calculated for each decision given the evidence. To represent variables that change over time, it is possible to use a time-sliced network such that each time-slice corresponds to a time point. A DDN is used for the states, preferences and the decisions that change over time. Fig. 8 shows the structure of a general DDN. To model the effectiveness of redeployments and reconfiguration over time, the decisions can be modeled using a DDN where each time slice contains an action taken by the system. Utility functions can be used to assign priorities to different QoS requirements. The random variables associated with the chance nodes in a DDN can represent the QoS requirements for all the possible actions.

| Component | CPU load | Communication |
|---|---|---|
| Accelerometer | 8.09 ± 1.3 ms | 5.5 ± 0.0 kB/sec |
| Low-Pass Filter | 57.9 ± 2.1 ms | 5.5 ± 0.0 kB/sec |
| Magnitude Filter | 18.2 ± 1.5 ms | 1.8 ± 0.0 kB/sec |
| Peak Detector | 14.9 ± 9.7 ms | 0.5 ± 0.4 kB/sec |
| Step Detector | 5.12 ± 4.8 ms | 0.1 ± 0.1 kB/sec |
| Fast Fourier | 5590 ± 108 ms | 2.5 ± 0.0 kB/sec |
| High-Pass Filter | 197 ± 8.2 ms | 5.0 ± 0.0 kB/sec |
| Signal Magnitude Area | 51.7 ± 3.4 ms | 2.0 ± 0.0 kB/sec |
| Fall Detector | 15.1 ± 9.1 ms | 2.0 ± 0.1 kB/sec |

Table 2

Performance benchmark of the individual components on the SunSPOT sensor

#### 5.2.2. Using the design-time meta data

Consider the structure of DDNs shown in Fig. 8. The chance nodes represent the Pareto optimal configurations, i.e. the states of the system. The evidence nodes are the different runtime situations based on the context of the user and the parameters of the system. The utility nodes are derived using the reconfiguration cost matrix. The decision nodes are the actual reconfiguration decisions.

## 6. Experimental evaluation

We will demonstrate the feasibility of our approach with both use cases mentioned earlier. These simple deployment scenarios allow deployment compositions on three different platforms: *Smart Object*, *Smart Mobile* or *Smart Server*.

- *Smart Object*: We use a SunSPOT development board (400MHz ARM 926ej-S processor with 1MB RAM and 8MB flash memory) and a resource constrained Raven Wireless kit (Atmel 8-bit AVR RISC-based microcontroller at 16MHz with 16KB SRAM and 128KB flash memory).
- *Smart Mobile*: All mobile application benchmarking results (including the Smart Lens ones) were obtained on a HTC One X smartphone with a 1.5 GHz Quad Core ARM Cortex processor.
- *Smart Server*: Our infrastructure runs on a pool of virtualized machines on top of 10 desktop machines with 8GB and a 3GHz multi-core CPU running a 64-bit edition of Ubuntu Linux 12.04.

### 6.1. Design time: Profiling

We profile the components under different deployment and configuration scenarios with an objective to optimize the CPU load and the network communication costs.

#### 6.1.1. Smart Object

The results of the step counting profiling on the SunSPOT sensor are shown in Table 2. Note that for the *Accelerometer, Low-pass filter* and *Magnitude filter* components there is little to no communication variability because the amount of data output is fixed and depending on the sampling rate of the accelerometer.

We were not able to test all the components on the Raven sensor, but those that were ported ran about 50 times slower compared to the SunSPOT.

#### 6.1.2. Smart Mobile

We have similar Android-based implementations for the smartphone, and depending on the hardware being used, we see a computational speed-up with a factor ranging between 20-60 (depending on the number of cores being used and their clock frequency).

For the second use case on energy awareness, we implemented a simple prototype using the Vuforia library. Our profiling experiments showed that the continuous processing of the camera preview (to recognize various artifacts) caused a continuous CPU load on the smart mobile between 40 and 45%. Due to the dependency of the camera component, and the fact that the Augmented Reality component is based on a native library, we cannot decompose the feature extraction and processing subcomponents for distributed deployment. However, reconfiguration of the component can still influence the computational load on the system (e.g. by changing the frame rate and frame size of the camera, the number of features per target image, and the maximum number of targets to be detected and tracked concurrently).

#### 6.1.3. Smart Server

For server side deployments, the computational cost of the step counting algorithms is negligible, and only becomes important if the algorithms are being executed for a large population.

Both use cases have the semantic localization in common. The first component takes raw data from various beacons as input (e.g. the signal strengths and/or fingerprints of WiFi access points) to compute a coordinate-based position, whereas the spatial semantic reasoner translates these coordinates into a semantically meaningful location. Due to the computational complexity, the first component cannot be run on the sensor (i.e. a *Smart Object*), but both the *Smart Mobile* and *Smart Server* are possible deployment options.

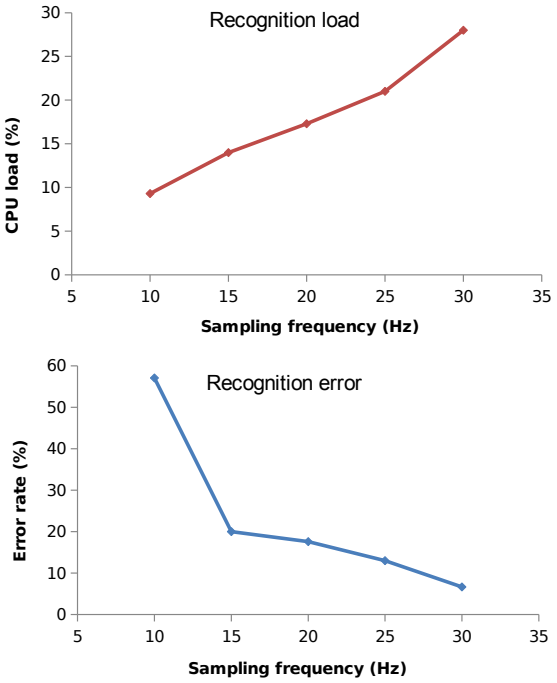


Fig. 9. Sampling rate vs. recognition accuracy and performance on the SunSPOT

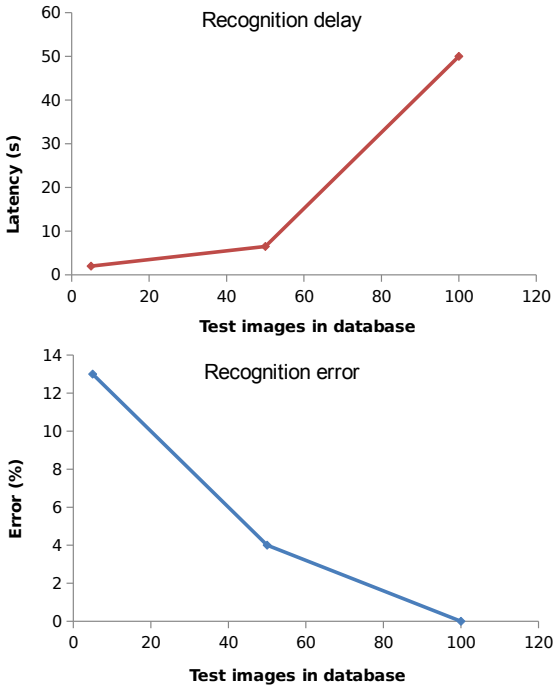


Fig. 10. Sample size in database vs. recognition accuracy and performance on the smartphone

However, the second component builds upon a heavy-weight semantic reasoner for which a *Smart Server* deployment is the only realistic option.

### 6.2. Design time: Pareto exploration

We analyzed for each component on each platform the trade-offs of sampling frequency against:

1. Recognition accuracy
2. Computational complexity

Fig. 9 depicts the results of these trade-off analyses for the step-counting components (features and feature classifiers) all running on the SunSPOT sensor. The figure shows two trade-offs of interest, i.e. (1) *recognition rate*, to compare different algorithms and configurations (e.g. size of sliding window and use of certain filters), and (2) *performance impact*, to decide which components to deploy and on which platform (computation vs. communication trade-off). Lowering the accelerometer sampling rate from 50Hz to 15Hz decreases the CPU time, communication and energy consumption of the activity recognition components, but increases the recognition error rate (i.e. inaccurate detection of steps). Similar trade-offs – not shown here – investigate scenarios with all the processing done on a gateway and intermediate deployments to compare the network overhead and power consumption vs. the sampling frequency. These kind of trade-offs help us to find Pareto-optimal deployments and configurations for activity recognition.

For the Smart Lens use case, the trade-offs we explored are between the recognition rate for target objects and the latency for recognition with respect to the number of test samples in the database. For each object to be recognized, the number of similar test samples increases from 5 to 100. The objective is for the camera of the smartphone to recognize a target object against the test samples in the target database. Fig. 10 depicts the results of these trade-off analyses for the Smart Lens components (features and feature classifiers) running on the smartphone.

### 6.3. Design time: Reconfiguration costs

In another exploration experiment, we compared the use of dedicated components on the sensors with a deployment that used the script engine. The advantage of a reusable scripting engine is that the smaller sensors like the Raven are limited in the number of components that they can deploy. Furthermore, with a

reusable scripting engine the redeployment costs for new configurations can be reduced significantly. These results are shown below.

Table 3 illustrates average costs for some very frequently occurring reconfigurations. The memory costs are measured by monitoring the stack and heap footprints and the energy costs are calculated by using the power consumption values for the CPU and radio provided in the datasheets. We observe that deploying the component takes a much more significant amount of energy and memory as compared to the other two reconfigurations.

| Reconfiguration type | Energy (mJ) | Memory (Bytes) |
|---|---|---|
| Deploy component | 0.2 | 256 |
| Script activation | 0.0 | 20 |
| Change frequency | 0.0 | 0 |

Table 3
Average costs of common reconfigurations on the Ravens

Even though components are more expensive to deploy in terms of energy and require more memory, the runtime overhead of dedicated components is much lesser compared to the generic script engine. Figure 11 shows the trade-off of using a dedicated component versus a script engine. We can see that for less than 45 messages the script engine is the optimal choice because the price of deploying a component is not recovered by the differences in the overhead. However, for more than 45 messages deploying a dedicated component becomes the optimal choice.

### 6.4. Runtime: Dynamic decision networks

At design time, we profile the components under particular circumstances and configurations. However, we cannot be sure that the costs and benefits derived from the Pareto exploration will be the same under all runtime circumstances. We therefore use Dynamic Decision Networks as a way to observe and learn whether any decision made based on the Pareto-fronts delivers what was promised. If not, we tune at runtime the utility of the deployment or reconfiguration decision based on the latest observations.

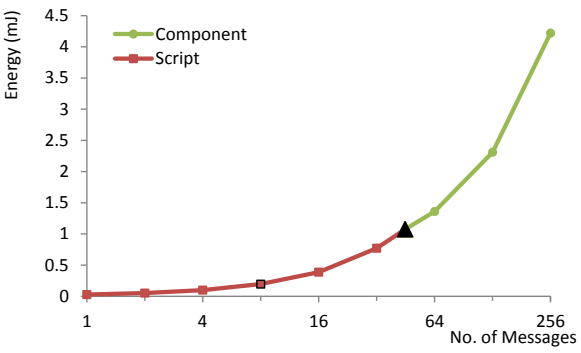


Fig. 11. Energy consumption verses number of samples for Ravens

In Fig. 12, we illustrate the benefits of using DDNs as a way to learn the deployment trade-offs under changing circumstances. This example represents the utility of 2 sampling rate configurations of the accelerometer: 50Hz and 10Hz. This utility is a weighted cost function based a.o. on the signal magnitude area (SMA). The SMA has a low value while idle, and a high value while walking or running. The figure depicts a user first moving around (at time slice 0), with the 50 Hz accelerometer sampling rate having the highest utility during the first two time slices. At that moment, the user sits down and the utility of the 50Hz configuration drops because of a low SMA, whereas the utility of the 10Hz configuration increases for the same reason.

This behavior can be explained as follows. To do step counting, a sampling frequency of 50Hz is necessary to accurately count the number of steps. However, with high sampling frequencies come high processing costs. As soon as the person becomes idle, this configuration wastes a lot of resources and hence incurs a low utility. In this setting, the 10Hz configuration is much more appropriate. If at some point, however, the user starts moving around again, the utility of the 10Hz configuration would drop, as steps are not detected properly anymore at this low sampling rate.

## 7. Conclusions and future work

Finding the best strategy to deploy and configure context-aware component-based applications in a smart environment with dynamic and heterogeneous resource availability is far from straightforward, especially if the context in which the application is being consumed becomes an integral part of how to best optimize the application. The dynamic deployment of software components in an Internet of Things (IoT) ecosystem has to take into account the resource characteristics – including processing power, bandwidth, battery life and connectivity – of the application components and the deployment platforms.

We have demonstrated through various context-aware applications for intelligent environments that re-

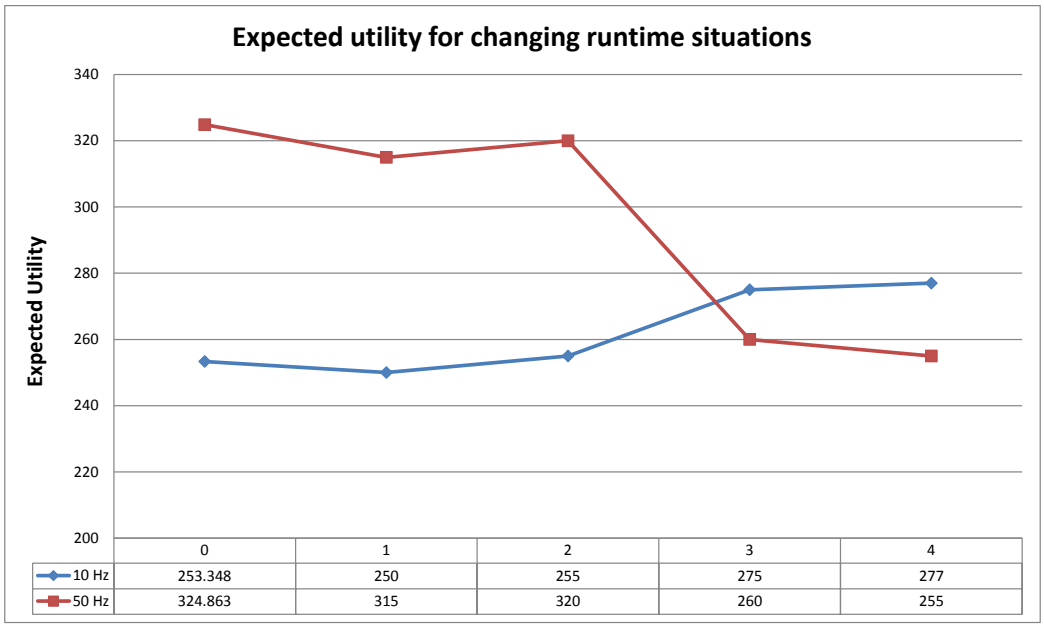


Fig. 12. Expected utility for changing runtime situations

source and performance trade-offs complicate the the decision of deploying a context-aware application (or some of its components) on either the sensor, the mobile or in the cloud.

In this paper, we have shown how a modular design philosophy can enable more optimal deployments of IoT applications for intelligent environments. We presented our methodology to inspect and learn the trade-offs of different deployment schemes of IoT applications in order to autonomously optimize their configuration at runtime. Based on QoS requirements and contextual dependencies, we can exploit the component-based design of the IoT applications to dynamically configure, compose and deploy these components.

At design time, our approach first starts with profiling and benchmarking these components on different deployment platforms. This exploration for Pareto-optimal solutions produces information that enables us to find trade-offs for a distributed deployment in terms of the performance impact as well as the cost/benefit of any reconfiguration or change in a particular component deployment. The overall aim of our work is to intelligently automate the distributed deployment and configuration of the components across the three types of platforms with different capabilities and resource availabilities (Smart Objects, Smart Mobiles and Smart Servers).

Our approach is complemented with a runtime phase that autonomously adapt the deployment and configuration towards changing operational circumstances:

1. We use annotated component graphs to model application compositions at design time.
2. Pareto-curves are used to represent the optimization options for each (type of) platform. The resource optimization objectives are chosen w.r.t. to the QoS requirements and trade-offs between computation and communication.
3. We use Dynamic Decision Networks for the runtime configuration and deployment to achieve the self-optimization capabilities of the system.

Our experiments show that with this combined approach, our framework is able to learn deployment trade-offs of smart applications for intelligent environments and capable of learning from earlier deployment or configuration mistakes to better adapt to the setting at hand.

As future work, we will focus on broadening our methodology to validate more complex deployment scenarios. Furthermore, a systematic approach to study how the values of the probabilities change over time would give us better insights on how they impact alternative deployment and configuration decisions. Last but not least, integrated tool support that embraces our

whole methodology – from profiling, Pareto exploration up to runtime support for Dynamic Decision Networks – would be certainly very helpful as the current tool's support is fairly limited.

## Acknowledgments

This research is partially funded by the Research Fund KU Leuven and the FP7 BUTLER[5] project.

## References

[1] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook, Discovering frequent user–environment interactions in intelligent environments, *Personal Ubiquitous Comput.*, **16**(1):91–103, January 2012.

[2] D. Bandyopadhyay and J. Sen, Internet of things: Applications and challenges in technology and standardization, *Wireless Personal Communications*, **58**:49–69, 2011.

[3] N. Bencomo and A. Belaggoun, Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks, In *Requirements Engineering: Foundation for Software Quality*, pages 221–236, Springer, 2013.

[4] Z. Bhatti, N. Miniskar, D. Preuveneers, R. Wuyts, Y. Berbers, and F. Catthoor, Memory and communication driven spatio-temporal scheduling on mpsocs, In *Integrated Circuits and Systems Design (SBCCI), 2012 25th Symposium on*, pages 1–6, 30 2012-Sept. 2.

[5] Z. Bhatti, D. Preuveneers, Y. Berbers, N. Miniskar, and R. Wuyts, Samosa: Scratchpad aware mapping of streaming applications, In *System on Chip (SoC), 2011 International Symposium on*, pages 48–55, 2011.

[6] Y. Censor, Pareto optimality in multiobjective problems, *Applied Mathematics and Optimization*, **4**:41–59, 1977.

[7] S. Chen, J. J. Lukkien, and P. H. F. M. Verhoeven, Context-aware resource management for secure end-to-end qos provision in service oriented applications, *JAISE*, **3**(4):333–347, 2011.

[8] E. Commission and J. Centre, *Active Ageing and Independent Living Services: The Role of Information and Communication Technology*, Dictus Publishing, 2012.

[9] D. J. Cook and L. B. Holder, Sensor selection to support practical use of health-monitoring smart environments, *Wiley Int. Rev. Data Min. and Knowl. Disc.*, **1**(4):339–351, July 2011.

[10] D. Deb, M. M. Fuad, and M. J. Oudshoorn, Achieving self-managed deployment in a distributed environment, *J. Comp. Methods in Sci. and Eng.*, **11**(3, Supplement 1):115–125, August 2011.

[11] K. Deb, Multi-objective optimization, In E. K. Burke and G. Kendall, editors, *Search Methodologies*, pages 273–316, Springer US, 2005.

[12] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, *Wireless Communications and Mobile Computing*, 2011.

[13] N. Esfahani, K. Razavi, and S. Malek, Dealing with uncertainty in early software architecture, In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 21, ACM, 2012.

[14] N. Fenton and M. Neil, Making decisions: using bayesian nets and mcda, *Knowledge-Based Systems*, **14**(7):307–325, 2001.

[15] A. Filieri, C. Ghezzi, and G. Tamburrelli, A formal approach to adaptive software: continuous assurance of non-functional requirements, *Formal Aspects of Computing*, **24**(2):163–186, 2012.

[16] L. Guan, X. Ke, M. Song, and J. Song, A survey of research on mobile cloud computing, In *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science*, ICIS '11, pages 387–392, Washington, DC, USA, 2011, IEEE Computer Society.

[17] G. Holmes, A. Donkin, and I. Witten, Weka: a machine learning workbench, In *Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357 –361, nov-2 dec 1994.

[18] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, Resource provisioning for cloud computing, In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '09, pages 101–111, Riverton, NJ, USA, 2009, IBM Corp.

[19] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Cambridge University Press, 1993.

[20] M. Koehler and S. Benkner, Design of an adaptive framework for utility-based optimization of scientific applications in the cloud, In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 303–308, Washington, DC, USA, 2012, IEEE Computer Society.

[21] N. C. Krishnan, C. Juillard, D. Colbry, and S. Panchanathan, Recognition of hand movements using wearable accelerometers, *J. Ambient Intell. Smart Environ.*, **1**(2):143–155, April 2009.

[22] K. Kumar and Y.-H. Lu, Cloud computing for mobile users: Can offloading computation save energy?, *Computer*, **43**(4):51 –56, april 2010.

[23] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, Activity recognition using cell phone accelerometers, *SIGKDD Explor. Newsl.*, **12**(2):74–82, March 2011.

[24] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, The jigsaw continuous sensing engine for mobile phone applications, In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 71–84, New York, NY, USA, 2010, ACM.

[25] R. Marler and J. Arora, Survey of multi-objective optimization methods for engineering, *Structural and Multidisciplinary Optimization*, **26**:369–395, 2004.

[26] A. P. Miettinen and J. K. Nurminen, Energy efficiency of mobile clients in cloud computing, In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 4–4, Berkeley, CA, USA, 2010, USENIX Association.

[27] S. N. Z. Naqvi, A. Ramakrishnan, D. Preuveneers, and Y. Berbers, Walking in the clouds: deployment and performance trade-offs of smart mobile applications for intelligent environments, In *Proceedings of the 9th International Conference on Intelligent Environments (IE13)*, pages 212–219, IEEE

[5] http://www.iot-butler.eu

Computer Society, July 2013.

[28] D. Preuveneers and Y. Berbers, Mobile phones assisting with health self-care: a diabetes case study, In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 177–186, New York, NY, USA, 2008, ACM.

[29] D. Preuveneers and P. Novais, A survey of software engineering best practices for the development of smart applications in ambient intelligence, *J. Ambient Intell. Smart Environ.*, **4**(3):149–162, August 2012.

[30] H. Qi and A. Gani, Research on mobile cloud computing: Review, trend, and perspectives, *CoRR*, **abs/1206.1118**, 2012.

[31] A. Ramakrishnan, S. N. Z. Naqvi, Z. W. Bhatti, D. Preuveneers, and Y. Berbers, Learning deployment trade-offs for self-optimization of Internet of Things applications, In *Proceedings of the 10th International Conference on Autonomic Computing, ICAC 2013, ICAC '13, the 10th International Conference on Autonomic Computing, San Jose, CA, U.S.A., 26-28 June 2013*, pages 213–224, ACM, June 2013.

[32] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, Activity recognition from accelerometer data, In *Proceedings of the 17th conference on Innovative applications of artificial intelligence - Volume 3*, IAAI'05, pages 1541–1546, AAAI Press, 2005.

[33] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*, volume 74, Prentice hall Englewood Cliffs, 1995.

[34] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, The case for vm-based cloudlets in mobile computing, *IEEE Pervasive Computing*, **8**(4):14–23, October 2009.

[35] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, Using constraint programming to manage configurations in self-adaptive systems, *Computer*, **45**(10):56–63, 2012.

[36] H. Sun, V. D. Florio, N. Gui, and C. Blondia, Promises and challenges of ambient assisted living systems, In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ITNG '09, pages 1201–1207, Washington, DC, USA, 2009, IEEE Computer Society.

[37] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, Vision and challenges for realising the internet of things, *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.

[38] S. Szewcyzk, K. Dwan, B. Minor, B. Swedlove, and D. Cook, Annotating smart environment sensor data for activity learning, *Technol. Health Care*, **17**(3):161–169, August 2009.

[39] H. J. ter Horst and A. Sinitsyn, Structuring reasoning for interpretation of sensor data in home-based health and well-being monitoring applications, *JAISE*, **4**(5):461–476, 2012.

[40] G. Tesauro, Online resource allocation using decompositional reinforcement learning, In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, AAAI'05, pages 886–891, AAAI Press, 2005.

[41] G. Tesauro, Reinforcement learning in autonomic computing: A manifesto and case studies, *IEEE Internet Computing*, **11**(1):22–30, January 2007.

[42] G. Tesauro and J. O. Kephart, Utility functions in autonomic systems, In *Proceedings of the First International Conference on Autonomic Computing*, ICAC '04, pages 70–77, Washington, DC, USA, 2004, IEEE Computer Society.

[43] D. Vengerov, A reinforcement learning approach to dynamic resource allocation, *Eng. Appl. Artif. Intell.*, **20**(3):383–390, April 2007.

[44] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. Jubert, M. Mazura, M. Harrison, M. Eisenhauer10, et al., Internet of things strategic research roadmap, *Internet of Things: Global Technological and Societal Trends*, page 9, 2009.

[45] A. Y. Yang, R. Jafari, S. S. Sastry, and R. Bajcsy, Distributed recognition of human actions using wearable motion sensor networks, *J. Ambient Intell. Smart Environ.*, **1**(2):103–115, April 2009.

[46] E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001, International Center for Numerical Methods in Engineering.